文章编号:1671-4229(2022)01-0018-09

分布式CNN中基于参数优先级的细粒度调度算法

姜 珊[a],程志超[a],曾荣飞[b],黄 敏[c],王兴伟[a*]
(东北大学 a. 计算机科学与工程学院; b. 软件学院;
c. 信息科学与工程学院, 辽宁 沈阳 110819)

摘 要:随着卷积神经网络模型日益复杂,训练数据类型更加丰富,数据量急速增长,单一机器已经无法满足模型训练的需求,分布式CNN成为新的解决方法。在参数服务器架构下,分布式CNN模型训练会产生大量的通信数据,可能会在每次迭代后产生突发流量,从而造成网络阻塞。在TensorFlow这种用图来表示计算的平台中,节点之间接收参数的顺序是随机的。由于数据流模型可能有多个可行的遍历,为了避免由于参数未接收完毕而造成的计算阻塞,尽量缩短迭代延迟,文章设计了基于计算图和参数优先级的细粒度调度机制,提出一种启发式算法,实现对计算图中所有节点强制执行最优顺序,通过改善通信和计算的重叠,实现细粒度的参数调度,并对VGG16和ResNet32进行测试对比。实验结果表明,使用细粒度的参数调度算法能够使迭代时间缩短7%~22%左右,从而缓解网络瓶颈,提高分布式CNN模型训练性能。

关键词:卷积神经网络;计算图;参数传输;优先级
中图分类号:TP 301.6 **文献标志码**:A

Fine grained scheduling algorithm based on parameter priority in distributed CNN

JIANG Shan[a], *CHENG Zhi-chao*[a], *ZENG Rong-fei*[b], *HUANG Min*[c], *WANG Xing-wei*[a*]
(a. School of Computer Science and Engineering; b. School of Software;
c. School of Information Science and Engineering, Northeastern University, Shenyang 110819, China)

Abstract: With convolutional neural network models becoming more and more complex, the types of training data become more and more abundant, and the amount of data increases rapidly, a single machine has been unable to meet the needs of model training, and distributed CNN has become the new solution. Under a parameter server architecture, distributed CNN model training will produce a large amount of communication data, which may generate burst traffic after each iteration, resulting in network congestion. In TensorFlow, a graph based computing platform, the order of receiving parameters between nodes is random. Because the data flow model may have multiple feasible traversals, in order to avoid the calculation blocking caused by the incomplete receiving of parameters and minimize the iteration delay, this paper designs a fine-grained scheduling mechanism based on computation graph and parameter priority. A heuristic algorithm is proposed to enforce the optimal order of all nodes in the computational graph. By improving the overlap of communication and computation, fine-grained parameter scheduling is realized. Vgg16 and ResNet32 are tested and compared. The experimental results show that the fine-grained parameter scheduling algorithm can shorten the iteration time by about

基金项目:国家自然科学基金资助项目(62032013);辽宁省"兴辽英才计划"资助项目(XLYC1902010)
作者简介:姜珊(1996—),女,硕士研究生. E-mail:jiangshan_mail@163.com
* 通信作者. E-mail:wangxw@mail.neu.edu.cn
引文格式:姜珊, 程志超, 曾荣飞, 等. 分布式CNN中基于参数优先级的细粒度调度算法[J]. 广州大学学报(自然科学版),2022,21(1):18-26.

7% ~22% , so as to alleviate the network bottleneck and improve the training performance of distributed CNN model.

Key words: convolutional neural networks; calculation graph; parameter transmission; priority

由于机器学习(Machine Learning,ML)框架和平台在开发上具有很大的灵活性,同时,越来越丰富的数据集以及逐渐完善的高性能计算体系,也使得人工智能领域得以迅猛发展。随着机器学习模型愈发复杂,训练深度逐渐加深,训练模型的计算成本超出了单个机器的负荷能力,因此,分布式机器学习(Distributed Machine Learning,DML)成为短时间内完成大规模模型训练的有效解决方案之一。卷积神经网络(Convolutional Neural Networks,CNN)作为 ML 的一种代表性算法,主要用于图像特征提取,已经成功应用在图像识别和自然语言处理领域。随着计算机软硬件的不断发展,网络模型日益复杂,数据类型愈加丰富,数据量急速增长,单一机器已经无法满足需求,分布式 CNN 成为新的解决方法。

在分布式 CNN 模型训练中,每次迭代都是计算节点先接收到更新后的参数,然后根据各自的训练数据子集进行计算梯度,最后,聚集不同节点的梯度以更新模型参数。因此,随着 DML 集群规模的增长,通信可能成为制约分布式 CNN 模型训练速度的瓶颈之一。同时,随着计算机硬件加速器的快速发展,如 GPU 和 FPGAs,频繁的参数/梯度交换很容易使网络通信成为瓶颈,从而降低 DML 的训练性能[1-2]。

当前系统中的数据传输时间都是在执行过程中确定的,在这期间并没有考虑参数计算和通信重叠带来的影响。TensorFlow 是一种用图来表示计算的平台。在数据并行模式下,每个计算节点都有一套完整的模型副本和部分数据子集,即数据集被分割成多个子集。每次迭代由参与的计算节点使用相同的计算图进行处理,每次迭代通常持续几毫秒到几秒钟。在每次迭代结束时,参数服务器会聚合所有节点的梯度来更新模型参数,在此过程中服务器会交换大量数据,这种通信开销对系统的吞吐量有很大影响,也限制了模型的可扩展性[3-4]。

分布式 CNN 的迭代时间取决于计算时间和通信时间。每次迭代开始时,计算节点会从参数服务器中获取参数,但是接收到的参数并不会同时被使用,而是根据底层计算图中的依赖关系进行使用[5]。由于计算节点之间接收参数的顺序是随机的,数据流模型可能有多个可行的遍历,可能造成因参数未接收完毕而带来的计算阻塞。因此,如果可以找到一个最优的参数执行顺序,会有助于改善计算和通信之间的重叠度,进而缩短迭代时间,加快模型训练。

1　相关工作

随着 DML 集群规模的不断增长,多个节点和服务器之间的通信可能会制约模型的训练速度。在典型的随机梯度下降方法中,直接获得解析解是十分困难且不易实现的,需通过迭代精化不断逼近最优解。因此,在每次迭代过程中,每个计算节点首先从服务器端获取参数,根据各自的训练数据来计算梯度,然后将梯度推送至服务器,参数服务器将聚集所有计算节点的梯度以更新模型参数[6]。随后计算节点再一次从服务器端获取更新后的参数进行下次迭代。

在分布式 CNN 中,大量且频繁的参数/梯度交换很容易造成网络阻塞,从而降低跨机器分布式模型训练的性能。为了应对昂贵的参数同步成本,计算机相关硬件加速器逐渐发展,比如 RDMA 这种高速且低延迟的网络技术以及其他一些计算机硬件加速器被不断开发,用来加速神经网络模型训练。目前,有很多研究工作是针对如何加速分布式 CNN 模型训练,进而提高训练性能展开的。从多个不同的角度进行分析,研究出了许多有效的解决方案。除了一些硬件加速的研究,在模型和算法方面,模型压缩和参数压缩等是较为普遍的做法,比如网络剪枝、模型量化、低秩估计以及模型蒸馏。本文重点研究在网络层面分布式 CNN 模型训练中,如何通过改善计算和通信之间的重叠,即实现计算和通信的细粒度并行,将通信开销尽可能地隐藏在计算之后。

计算图是 CNN 模型训练的前提,无论是静态构建还是动态构建,在常用的模型副本(即数据并行训练的模式)中,输入数据被分割,并由参与计算的节点使用相同的计算图进行处理,每次迭代通常持续几毫秒到几秒钟。在每次迭代结束时,服务器会和所有计算节点交换大量与参数更新相关的数据。这种通信开销对系统的吞吐量有很大影响,也限制了其可扩展性[7]。在耗时较长的模型训练中,即使通信开销只稍微改善一点点,训练时间也可以缩短几个小时。之前的研究工作试图通过在所有计算节点中执行相同的参数传输顺序来解决这个问题。然而,这些研究工作是针对早期系统的模型结构,在大规模机器学习 TensorFlow 架构中[8],这是一个不小的挑战。

本文提出的启发式算法侧重于深度学习框架中的

网络优化,在参数服务器架构下采用模型副本模式进行模型训练。启发式算法带来的性能改进主要在2个方面:①提高网络吞吐量;②以计算节点为训练对象,强制排序可以缩短迭代时间。虽然分散聚合方式(如all-reduce和Horovod[9])在高性能网络中获得了越来越多的关注,但本文没有针对这类系统,而是基于参数服务器架构下进行研究。

在深度学习系统中,至关重要的通信成本随着横向扩展而增加。在这种情况下,当通信时间小于或等于计算时间,GPU利用率相对较高。此外,通信和计算的有效重叠对于提高吞吐量也很关键。目前已经提出了3种技术来提高系统性能:①增加计算时间。可以通过增加批量来增加计算时间相对于通信时间的比例[10],但是这种方案会降低模型训练的准确率[11],需要额外的校正机制,在资源限制下可能无法普遍适用[12-13];②减少通信时间。可以通过优化机器学习算法以降低通信成本[14],或者通过降低参数精度[15]来降低通信开销;③尝试改善计算和通信之间的重叠情况。例如,文献[5]将卷积层参数和全连接层参数赋予不同优先级,在整个网络结构中,卷积层参数始终优先于全连接层参数进行传输。由于每次CNN模型迭代都是先进行卷积操作、激活函数以及池化操作等,最后经过全连接层,因此,通过优先发送卷积层参数可以改善计算和通信的有效重叠。但这种参数调度是粗粒度的,当模型较为简单时,会获得一定的性能提升;当模型深度较大时,存在多个卷积层,无法对参数进行有效调度。文献[11]在分布式深度神经网络中通过对参数进行切片,并根据深度神经网络的分层结构为它们指定优先级[16],也就是第一层为最高优先级,第二层为次高优先级,以此类推。每次都是从待执行队列中选取优先级最高的参数,被首先发送出去。然而这种解决方案只关注每个计算节点中参数的传输顺序,忽略了在网络结构中的整体情况,从而不能很好地协调来自不同计算节点的参数。在大规模DML中会有多个计算节点向服务器交换参数,网络中存在大量流量,此时这种终端优先级参数调度将不再有效。

本文提出一种细粒度的参数传输调度机制,通过对底层计算图的关键路径进行分析来获得参数传输的近似最优调度。通过一种启发式算法完成参数的优先级排序,实现计算和通信的最大重叠。在网络传输方面采用基于优先级的流调度方案,进一步加速卷积神经网络的训练。参数优先级会在数据包报头中携带,使得参数在整个网络结构中都可被网卡和交换机等识别。基于计算图的细粒度参数调度,可以在多个计算节点之间以不同的顺序激活网络传输,实现所有计算节点都遵循相同的最优调度顺序。通过细粒度的参数调度机制,可以改善分布式CNN训练中通信和计算的重叠,此外,该机制还可以使用所有由ML框架支持的加速器/网络结构,可以移植,且不需要对ML框架进行修改。

2 卷积神经网络

2.1 CNN模型训练

在自然语言处理等领域中,文本和图像无法直接被机器识别,需要将文本和图像进行处理,转换为数值型数据再进行使用。CNN的核心是卷积层,卷积层包含多个大小不同的滤波器,输入数据经过卷积处理后完成特征的提取。输入数据经历卷积操作、激活函数以及池化操作后,最后进入全连接层完成输入数据的特征提取。CNN模型由于加入了卷积层和池化层,相比于传统的神经网络模型,增强了模型的泛化能力,支持更复杂的网络模型,特征提取的准确度也较高。

CNN具有分层结构(图1),主要包括卷积层、池化层、激活层和全连接层等,每一层都包含大量的神经元,相邻层的这些神经元相互连接[5]。一般来说,卷积层包含的参数较少,但需要进行的计算较多。

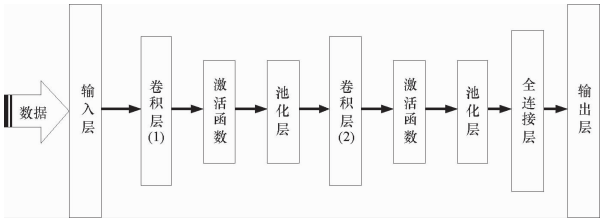

图1 CNN结构

Fig. 1 CNN structure

CNN的训练过程分为2个阶段。第一阶段是前向传播,第二阶段是反向传播阶段。训练过程(图2)分为5个步骤:

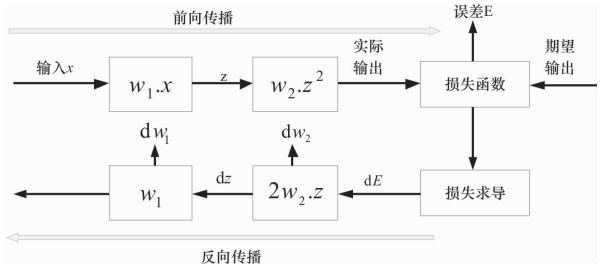

图2 CNN训练过程

Fig. 2 CNN training process

Step 1: 初始化权重和偏置;

Step 2: 前向传播;

Step 3: 求出误差;

Step 4: 当误差大于期望值时,将误差传回网络中;当误差等于或小于期望值时,结束训练;

Step 5: 更新权值,回到 Step 2 继续迭代。

2.2　PS 架构

随着 CNN 模型变得愈发复杂,计算工作量也相应增加。一方面模型参数过大,单机内存空间不足,需要采用分布式存储;另一方面训练数据过多,单机训练太慢,需要增加训练节点,来提高并发训练速度。因此,分布式 CNN 逐渐发展起来。在实践中最广泛采用的是参数服务器(PS)架构[6],并且得到了主流 DML 框架的良好支持,如 TensorFlow、PyTorch [17]等。

本文中设定分布式 CNN 以数据并行的方式在 PS 架构下训练,同步模式采用批量同步并行(BSP)。在每次迭代过程中,每个计算节点基于不同的数据子集进行迭代,最后由参数服务器聚合来自不同计算节点的梯度,更新模型参数用于下次迭代。也就是说,每次迭代过程包括从服务器获取参数、前向传播计算、反向传播计算、向服务器推送梯度、聚合梯度和更新参数。其中一些可以进行重叠,以达到更快的训练速度。参数服务器架构如图 3 所示。

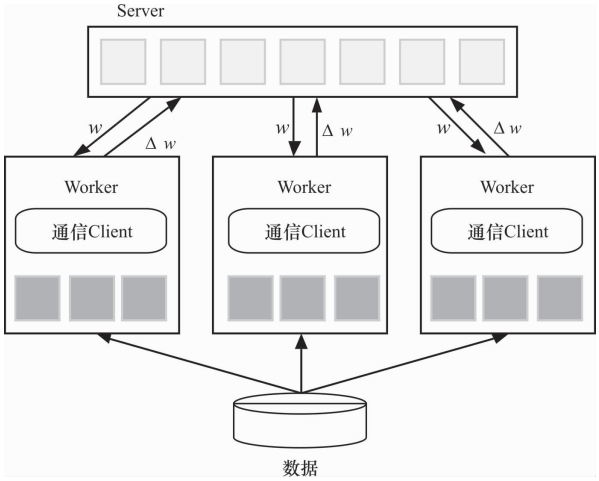

图 3　PS 架构

Fig. 3　PS framework

参数服务器架构主要包含 Server 和 Worker 2 个部分,训练的具体流程如下:

Step 1: 将训练数据分配给不同的 Worker;

Step 2: Worker 读取一个 mini batch 训练数据,从 Server 端获取最新的参数、计算梯度和推送梯度给 Server;

Step 3: Server 接收所有 Worker 上传的梯度,聚合后更新参数。

总迭代时间(T)、通信时间(t)和计算时间(C_t)之间的关系通常表示为 $T \leqslant t + C_t$,这是因为计算和通信可能重叠[18]。通信/计算比 ρ、重叠系数 α 和 GPU 利用率 U 的关系如下:

$$U = \frac{1}{1 + \rho - \alpha * \min(\rho, 1)} \tag{1}$$

当 $\rho < 1$ 时,通信时间小于总计算时间,表示此时 GPU 利用率较高,然而通信和计算的不良重叠会导致 GPU 利用率偏低。由于通信成本是随着横向扩展而增加的[19],当通信时间小于或等于计算时间时,GPU 利用率是较高的。因此,更好地优化通信和计算的有效重叠,可增加分布式 CNN 模型训练性能,提高吞吐量缩短迭代延迟。

2.3　计算图

计算图通常分为构建和执行 2 个阶段。计算图的组成部分见表 1。

表 1　计算图的组成

Table 1　Composition of computation graph

类型	描述	作用
Graph	计算过程	描述计算操作流程
Tensor	数据(边)	代表多维数组
Op	操作(节点)	具体执行操作
Session	会话	Graph 需要在 Session 启动

在分布式 CNN 模型训练过程中,每个 Worker 都有相同的模型副本。然而在参数服务器端有一个不同于 Worker 的计算图,该计算图中的参数涉及 5 个操作,分别是参数聚合、发送参数、接收参数、读取参数和更新参数。通常情况下,在参数服务器上的聚合、读取和更新是轻量级的。此时参数的传输是由 Worker 驱动的,每次迭代都是由参数服务器激活所有发送和接收操作,参数服务器负责控制网络传输,因此,在参数服务器中不会出现通信计算重叠的问题,只需考虑 Worker 中的计算图优化问题。

在 Worker 计算图中,所有获取参数操作(recv)都是根操作,推送参数操作都是叶操作。在某些情况下可以这样认为,recv 操作可能会阻碍计算图中分支计算,造成计算阻塞,延迟下次迭代的开始。本文的研究目标是为了达到更大的吞吐量和更小的延迟,结合 CNN 分层结构特点,通过更好地预测通信和计算重叠来缩短迭代时间,提高分布式训练性能。

计算图中的节点主要涉及 2 种类型的操作:①计算操作,如乘法、卷积等;②通信操作,如读取和更新。每

个参数都是独立读取和更新的。同样,每次迭代也有2个阶段:前向传播阶段和反向传播阶段。在前向传播中,会根据模型的输入计算损失函数;在反向传播阶段,基于计算的损失来更新模型参数。例如,图4为一个简单计算图,其中存在2种可能的参数传输方案。如果recv1在recv2和recv3操作完成之后执行,则缓解了计算阻塞;若首先执行recv1操作,会引发计算阻塞,导致迭代时间增加。因此,在分布式环境中,可以根据计算图中的依赖关系进行参数传输调度,优化计算和通信间的重叠程度,从而加快分布式CNN模型训练。

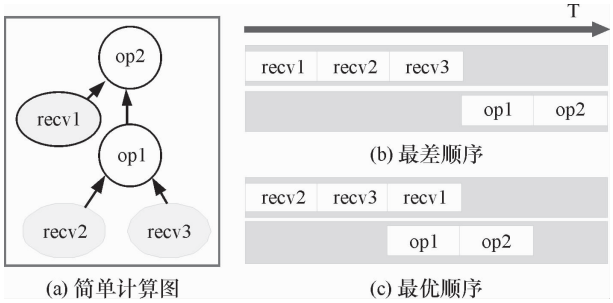

图4 不同参数传输顺序对计算阻塞的影响

Fig. 4 Influence of different parameter transmission sequence on computational blocking

在分布式CNN中,随着模型越来越复杂,参数也越来越多,需要将这些参数分散到多个节点进行存储和更新,主要包含2种方式:①In-graph模式,即将数据分发到1个节点上,然后再把数据分配给相应的节点,在数据量巨大的时候,这种模式会严重制约模型训练速度,但由于其不依赖于外部,因此,更容易访问,更容易在云环境中部署;②Between-graph模式,即数据分片分别保存在节点本地,每次迭代只需按照计算图来训练本地数据集,再将计算的梯度推送到参数服务器,大数据下的深度学习更适合这种模式。数据转换、传输和聚合被定义为标准数据流操作,其中,聚合被抽象为数据流模型中的单个操作,因此,将参数聚合实现为计算图的一部分。

3 参数传输调度机制

参数传输调度机制可以描述为如何在计算图中选择一个最优的操作执行顺序。其研究目标是改善通信和计算重叠,加快模型训练。该算法的输入包括2个部分:①资源关联图,即具有与每个操作相关联的资源标签计算图,其中,计算操作被分配给相应的计算资源,通信操作被分配给相应的通信信道;②预测时间表,即时间预测器预测给定操作的执行时间。对于计算操作表示计算时间,对于通信操作表示传输时间。该算法的输出是使迭代时间最小的可行操作拓扑顺序。

3.1 属性更新算法

分布式CNN模型训练中,Worker在每次迭代开始时从参数服务器获取参数,所有参数不会同时被使用,而是根据底层计算图中的依赖关系进行使用。然而一个特定的参数传输顺序可能有助于更快的计算,因此,确定参数传输的最佳调度对于减少计算阻塞(由计算图相关性决定),改善计算通信重叠及缩短迭代时间至关重要。

算法1 操作属性更新

//更新当前未完成接收参数操作的属性

1: function UPDATE(G)
2: for each op $\in$ 计算图 G do
3: $R \leftarrow \{\forall op \in G; op\ is\ recv\}$
4: end for
5: for each op $\in$ 操作集 R do
6: $op.graph \leftarrow$获取当前op的计算图
7: 遍历与当前op有数据依赖的节点
8: count$\leftarrow$节点个数
9: $op.count \leftarrow op.count + count$
10: end for
11: end function

本文侧重于深度学习框架中的网络优化,通过对底层计算图的关键路径分析来获得参数传输的近似最优调度,利用细粒度的调度来解决随机参数的传输问题,提高分布式CNN模型训练的性能。

进行算法设计时,优先考虑加速计算图中关键路径的传输,根据计算图中的节点依赖关系和每个操作的执行时间来分配优先级,对于计算阻塞更少的参数传输给予更高的优先级。通过对计算图中的所有recv操作节点进行优先级排序,选择通信和计算开销最小的recv操作,多次迭代直至遍历所有recv操作节点。添加recv操作优先级属性,通过遍历当前计算子图来实现。

3.2 优先级分配算法

优先级分配,即为了能够限制执行只接受一个有效集,考虑利用优先级来实现。优先级是分配给资源关联图中一个操作的正整数,优先级较高的操作被赋予较高的优先级。如果多个操作的相对顺序无关紧要,则分配相同的优先级。优先级仅指定在给定资源准备执行的队列中候选操作之间的相对顺序,并且所得到的顺序将仍然遵守由计算图指定的拓扑顺序。在分布式模型训练中,当一个资源需要从准备执行的队列中选择一个新的项目时,它会从包含最高优先级的队列中随机选择。

算法 2 优先级分配

//对所有 recv op 进行优先级排序

1: function ASSIGNPRIORITY(G, *Time*)

2:　Update(G)

3:　$\forall op \in G$;*op. time*←calculating time

4:　for R is not emptry do

5:　　Find the minimum op. time

6:　　Remove op from R;//从 R 中移除当前 recv op

7:　　*op. priority*←*op. count*;

8:　end for

9: end function

优先级分配算法主要是遍历计算图中的 recv op,对未完成的操作进行优先级排序,然后获得与当前 recv op 相关联的参数,将此时计数器数值赋予该参数为优先级数。该算法综合考虑计算图的依赖关系和实际执行时间,通过估计每个未完成 recv op 的未来通信和计算开销,对其进行排序,进而完成与之相依赖的参数优先级的分配。该算法由 2 个步骤组成,首先假设所有的操作都有相等的开销,只根据计算图中的节点依赖关系来分配优先级,对计算阻塞更少的传输给予更高的优先级。其次优先考虑加快关键路径上的传输,同时综合考虑每个操作的实际执行时间,通过时间预测器来估计每个操作的执行时间,结合计算图中的依赖关系,对最大化计算/通信重叠的传输进行优先级排序。

4　具体实现

4.1　参数优先级分配

首先获取资源关联图并评估每个操作的执行时间,结合计算图中节点的依赖关系,执行操作分配优先级算法,将此时的计数器数值赋予与该 recv op 相依赖的参数,作为该参数的优先级数。实现过程包括以下 2 个步骤:

(1)确定最佳顺序。为了最大化重叠程度,应尽可能早地激活通信操作,对 recv op 节点进行优先级排序,使用贪婪算法来获得参数更新的最佳顺序。在每一次迭代中,选择激活所需计算成本最小的参数,并且它所依赖的参数被标记为相同的优先级数。重复该过程,直到访问完所有 recv op。

(2)强制执行最优顺序。将与当前 recv op 相关联的参数赋予相同的优先级数,且始终携带在其数据包报头中,实现在整个网络结构中传输时,仍然以此优先级作为优先转发的依据。

4.2　网络传输

在训练开始期间,在不同的节点之间建立多个具有不同优先级的流,并生成通道表,同时为每一个流分配一个唯一的本地标识。当需要发送参数时,首先查找参数-优先级映射表,获得一个优先级编号,然后根据优先级编号查找通道表,获得一个合适的流进行发送参数。该参数被发送到网络结构中,在整个网络结构中,优先级标签总是在数据包报头中携带。当多个参数同时到达交换机时,交换机可以通过标签来区分它们,并根据优先级标签进一步转发它们。

网络参数调度可以在较为复杂的模型训练中使用,终端参数调度则不能发挥优势。例如,在 DML 集群中有多个计算节点,为了简化起见,在图中只显示了 2 个计算节点和 1 个参数服务器,图 5 为终端参数调度和网络参数调度之间的比较。

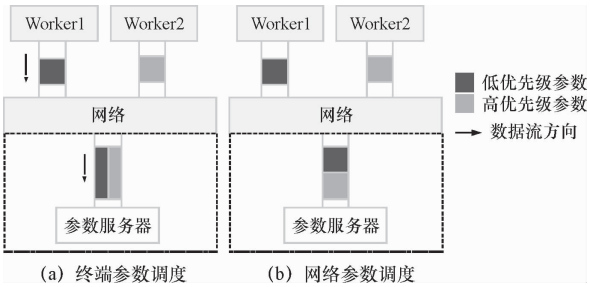

图 5　终端参数调度与网络参数调度的比较

Fig. 5　Comparison between terminal parameter scheduling and network parameter scheduling

对于网络调度,流的优先级数和参数的优先级数相匹配,即较高优先级的参数被分配较高优先级的流。当 Worker1 和 Worker2 分别发送一个参数,假设 Worker1 发送参数的优先级较低,Worker2 发送参数的优先级较高,对于网络调度,可以通过识别优先级标签来选择发送参数的顺序,即高优先级参数将被调度在低优先级参数之前发送;但终端调度只能保证参数在本地的执行顺序,无法满足在网络结构中的有效调度,利用终端调度的实际情况是 Worker1 上的低优先级参数和 Worker2 上的高优先级参数没有执行调度(此时终端调度不起作用),而是这 2 个参数平等地共享有限的带宽,Worker2 上的参数需要花费 2 倍于网络调度的时间才能到达服务器。这种情况还会随着计算节点数量的增加而恶化,因此,纵观整个网络结构,网络参数调度比终端参数调度更有扩展性,调度效率也更高。

4.3　实验分析

本文通过对 VGG16 和 ResNet32 等模型进行实验,分析提出参数传输调度方法的效率。实验环境为 Intel

(R) Core(TM) i7-7500U CPU @ 2.70 GHz 2.90 GHz、windows10、Tensorflow-gpu 2.2.0、Python 3.8.3 及 conda 4.9.2,数据集为 cifar10。

通过分析 VGG16 和 ResNet32 的参数分布(图6、图7),可以发现很多 CNN 模型的参数是偏斜分布的,即最后几层占据总参数的大部分。比如在 VGG16 模型中,全连接层中的参数分别占总参数的 80% 左右。因此,通过实现卷积层和全连接层的参数并行传输,优化参数传输的细粒度调度,才有可能加速分布式 CNN 的模型训练。

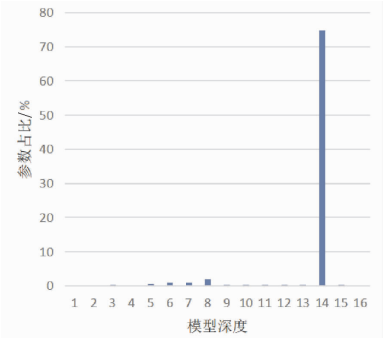

图6 VGG16 参数分布

Fig. 6 VGG16 parameter distribution

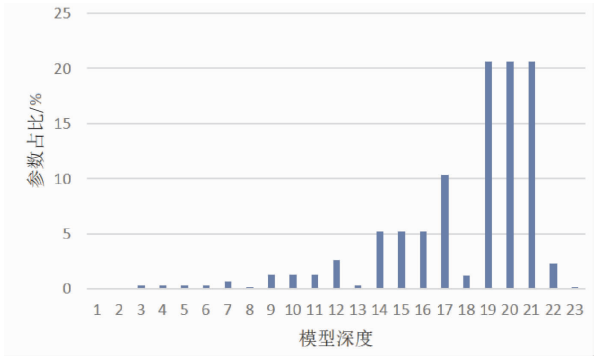

图7 ResNet32 参数分布

Fig. 7 ResNet32 parameter distribution

gRPC 为每个计算节点 - 参数服务器对提供一个通道,该对之间的所有传输都被发送到同一队列。在给定时刻,每个通道只能有一个传输处于活动状态。网络传输生命周期如图8所示,TensorFlow 中通过 gRPC 的网络传输涉及多个阶段。当接收方激活 recv op 时,它会向发送方发送一个传输请求,如果发送方的发送操作也处于活动状态,则参数传输由 gRPC 发起。此外,还定义一个度量调度效率,来衡量调度性能收益,评估启发式算法的质量。输入计算图、资源集和时间表,时间表给出了每个操作的执行时间。最长完成时间的上限是假设在执行期间的任何给定时刻仅使用一个资源来计算,即操作是顺序执行的;最长完成时间的下限是假设所有的资源总是被利用来计算。

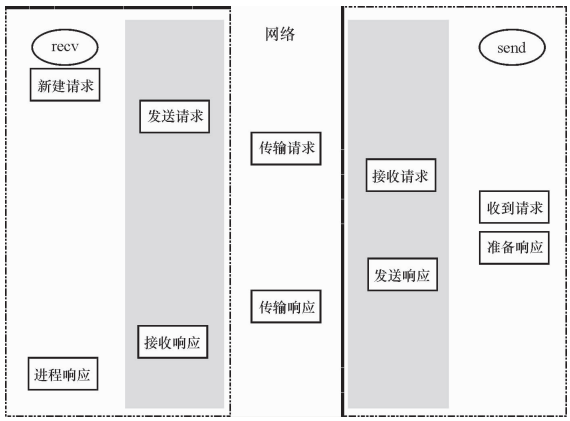

图8 网络传输生命周期

Fig. 8 Lifetime of a network transfer

实验使用的数据集 cifar10 是一个小型数据集,共有 60 000 张 32 * 32 大小的 RGB 彩色图片,共 10 个分类,训练数据和测试数据分别为 50 000 张和 10 000 张。目前的实验是生成一个 In-graph 数据流模型,其中,所有计算节点的数据流都表示在一个计算图中,然后对数据进行分区。此计算图的大小随着计算节点数量的增加而增加,这可能会增加大型图的流处理时间。接下来会使用 Between-graph 数据流模型,其中,每个计算节点的数据流模型都会单独生成,使这种细粒度的参数调度机制更具有扩展性,更适合大数据下的复杂模型。针对 VGG16 和 ResNet32 进行对比实验,引入启发式算法后的细粒度调度机制能够有效缩短迭代时间,但对于动态和可变模型,该机制目前还无法准确预测具有动态控制流的数据流模型或具有高度可变输入大小的模型时序,因此,不准确的预测会导致更长的迭代时间,这也是下一步研究需要解决的问题之一。

参数在计算节点和服务器之间进行传输和聚合,在每一次迭代中,计算节点从服务器获取参数,计算节点推送梯度到服务器,为了提高数据在网络上传输的网络利用率,可以通过参数计算和参数传输的并行执行及参数的细粒度流水线传输来实现。另一种避免这种网络利用不足的解决方案是将数据分割成几个部分,并行独立传输每个块。组块数称为算法深度。在这种情况下,当一个块在中央处理器上减少时,另一个块可以通过网络发送,即实现了跨各种块的网络传输和网络处理的流水线操作。

在参数传输和聚合阶段,需要根据计算图的依赖关系确定 recv 操作的近似最优执行顺序,且确保在本地模型中只有一个可行的执行顺序。首先确定最优执行顺序,确定参数的优先等级,尽可能早地激活计算阻塞更少的 recv 操作。针对每个 recv 操作尽可能添加少量的

操作属性,保证参数可以按照最优顺序进行更新/激活。先在一台机器上跟踪迭代执行 10 次,计算每个操作的执行时间,接下来,通过使用一个迭代贪心算法来寻找最优参数更新顺序。每次迭代中,在计算节点获取参数之前,需要评估完成计算操作所花费的总时间,然后选择激活当前时间成本最小的参数,将和该参数所依赖的计算操作标记为已完成。重复此过程,直到所有参数已访问更新。

最后是执行最优参数顺序,这是一个迭代过程,参数会按照前一阶段确定的最优顺序进行参数传输。在每一步中,都会找到所选参数直接或间接依赖的所有操作列表。为计算图中的每个 recv 操作维护一个计数器,遍历当前计算子图所有入度为零的操作,每遍历到一个与之依赖的操作,便将当前 recv 操作的计数器数值加 1,以此类推,直至遍历完当前计算子图。选择发送参数时,要先寻找与之相依赖的 recv 操作的计数器数值,将该数值赋予为参数的优先级数。这确保了在每个给定时间,只能执行目标参数更新所需的操作。但是向数据流模型添加额外的控制依赖不会改变计算图的底层逻辑,强制执行的最优顺序是计算图中可以执行的顺序之一。

由于数据流模型可能有多个可行的遍历,即数据流模型中计算操作的不同执行顺序都是有效的。计算图中的叶节点代表激活网络传输的 recv 操作。在参数服务器架构下,通常采用同步参数更新,在每次迭代后产生的突发流量容易造成网络阻塞,从而延迟参数传输。为了解决这个问题,基于计算图的细粒度参数调度,可以在多个计算节点之间以不同的顺序激活网络传输,实现所有计算节点都遵循最优的参数调度顺序。通过细粒度的参数调度机制,可以改善分布式 CNN 训练中通信和计算的重叠。此外,该机制还可以使用所有由 ML 框架支持的加速器/网络结构,可移植,且不需要对 ML 框架进行额外修改。

笔者在 TensorFlow 上实现了细粒度参数调度机制,其中,VGG16 模型训练数据集 cifar10,学习率设置为 0.000 1,batch_size 为 16,epoch 为 100 的训练结果如图 9 所示。ResNet32 模型同样训练数据集 cifar10,学习率为 0.000 1,batch_size 为 32,epoch 为 100 的训练结果如图 10 所示。

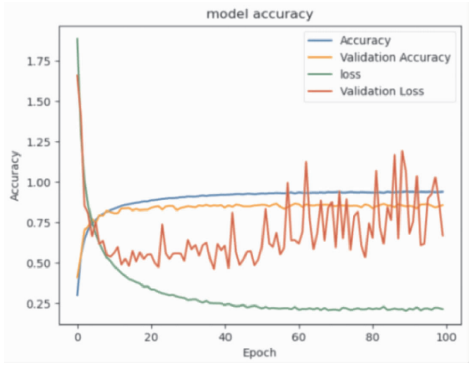

图 9　VGG16 模型训练

Fig. 9　VGG16 model training

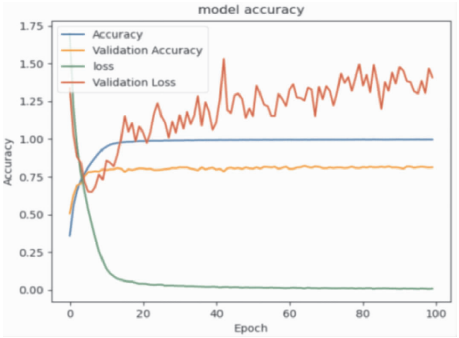

图 10　ResNet32 模型训练

Fig. 10　ResNet32 model training

实验分别在 VGG16、ResNet32 及 CNN + LSTM 3 种模型下进行对比分析,实验结果如表 2 所示。

表 2　CNN 模型训练迭代时间

Table 2　CNN Model training iteration time

Model	Accuracy	loss	Original iteration time	Optimization iteration
VGG16-cifar10	0.940	0.214 8	1 856 s/epoch	1 735 s/epoch
ResNet32-cifar10	0.997	0.009 7	3 598 s/epoch	2 814 s/epoch
CNN + LSTM	0.900	0.230 0	241 s	209 s

由表 2 可知,使用参数传输优化算法后,3 种模型都在不同程度上减少了迭代时延。特别在 CNN + LSTM 模型中,基准迭代时间是 241 s,引入启发式算法后迭代时间缩短为 209 s。在上述实验中,通过细粒度优化参数的传输顺序,迭代时间分别减少为之前的 93.48%、78.21% 和 86.72%,加快了 CNN 模型训练速度。

5　结　论

为了加快卷积神经网络的模型训练,本文设计了一种参数传输调度机制,通过一种启发式算法来实现基于计算图的参数传输机制,从而避免了由于参数未接收完

毕而造成的计算阻塞。为了缩短迭代延迟,首先对本地计算图中所有节点强制执行最优顺序,改善通信和计算的重叠,加快CNN的模型训练,从而提升分布式CNN的训练性能。通过对VGG16和ResNet32进行测试对比,实验结果表明,使用启发式算法后的CNN模型能够在不同程度上缩短训练迭代时间,迭代时间缩短了7%~22%左右。但对于动态和可变模型,该机制目前还无法准确预测具有动态控制流的数据流模型或具有高度可变输入大小的模型时序,因此,不准确的预测会导致更长的迭代时间,这也是下一步研究需要解决的问题之一。该算法在实际参数服务器架构中应用时,网络链路、参数同步方式、参数分片等情况,还有待进一步研究和优化。

参考文献:

[1] Cui H G, Zhang H, Gregory G, et al. GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server[C]//Eleventh European Conference on Computer Systems. New York: ACM, 2016: 1-16.

[2] Zhang H, Zheng Z, Xu S, et al. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters[C]//USENIX Annual Technical Conference. Berkeley:USENIX Association, 2017: 181-193.

[3] Wang S, Li D, Geng J, et al. Impact of network topology on the performance of DML: Theoretical analysis and practical factors[C]//IEEE INFOCOM 2019-IEEE Conference on Computer Communications. Piscataway: IEEE, 2019, doi: 10.1109/INFOCOM.2019.8737595.

[4] Sridharan S, Vaidyanathan K, Kalamkar D, et al. On scale-out deep learning training for cloud and HPC[EB/OL]. (2018-01-24)[2021-06-20]. https://arxiv.org/pdf/1801.08030.pdf.

[5] Wang S, Li D, Geng J. Geryon: Accelerating distributed CNN training by network-level flow scheduling[C]//IEEE INFOCOM 2020-IEEE Conference on Computer Communications. Piscataway: IEEE, 2020:1678-1687.

[6] Geng J, Li D, Cheng Y, et al. HiPS: Hierarchical parameter synchronization in large-scale distributed machine learning[C]//Proceedings of the Workshop on Network MeetsAI & ML 2018. New York: ACM, 2018, doi:10.1145/3229543.3229544.

[7] Alistar D, Grubic D, Li J, et al. QSGD: Communication-efficient SGD via gradient quantization and encoding[EB/OL]. (2017-12-06)[2021-06-20]. https://arxiv.org/pdf/1610.02132.pdf.

[8] Abadi M, Barham P, Chen J, et al. TensorFlow: A system for large-scale machine learning[C]//Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation. Berkeley:USENIX Association,2016: 265-283.

[9] Sergeev A, Balso M D. Horovod: Fast and easy distributed deep learning in TensorFlow[EB/OL]. (2018-02-15)[2021-06-20]. https://arxiv.org/pdf/1802.05799.pdf.

[10] Iandola F N, Moskewicz M W, Ashraf K, et al. FireCaffe: Near-linear acceleration of deep neural network training on compute clusters[C]//Computer Vision & Pattern Recognition. Piscataway:IEEE, 2016: 2592-2600.

[11] Keskar N S, Mudigere D, Nocedal J, et al. On large-batch training for deep learning: Generalization gap and sharp minima[EB/OL]. (2017-12-09)[2021-06-20]. https://arxiv.org/pdf/1609.04836.pdf.

[12] Goyal P, Dollár P, Girshick R, et al. Accurate, large minibatch SGD: Training ImageNet in 1 hour[EB/OL]. (2017-06-08)[2021-06-20]. https://arxiv.org/pdf/1706.02677v1.pdf.

[13] Akiba T, Suzuki S, Fukuda K. Extremely large minibatch SGD: Training ResNet-50 on ImageNet in 15 minutes[EB/OL]. (2017-11-12)[2021-06-20]. https://arxiv.org/pdf/1711.04325.pdf.

[14] Wen W, Xu C, Yan F, et al. TernGrad:Ternary gradients to reduce communication in distributed deep learning[C]//The 31st Conference on Neural Information Processing Systems. La Jolla: NIPS, 2017: 1508-1518.

[15] Courbariaux M, Bengio Y, David J P. BinaryConnect: Training deep neural networks with binary weights during propagations[EB/OL]. (2015-11-02)[2021-06-20]. https://arxiv.org/pdf/1511.00363.pdf.

[16] Jayarajan A, Wei J, Gibson G, et al. Priority-based parameter propagation for distributed DNN training[EB/OL]. (2019-05-10)[2021-06-22]. https://arxiv.org/pdf/1905.03960v1.

[17] Paszke A, Gross S, Chintala S, et al. Automatic differentiation in PyTorch[C]//The 31st Conference on Neural Information Processing Systems. La Jolla:NIPS, 2017.

[18] Hashemi S H, Jyothi S A, Campbell R H. TicTac: Accelerating distributed deep learning with communication scheduling[EB/OL]. (2018-03-08)[2021-06-20]. https://arxiv.org/pdf/1803.03288.pdf.

[19] Park J H, Kim S, Lee J, et al. Accelerated training for CNN distributed deep learning through automatic resource-aware layer placement[EB/OL]. (2019-01-17)[2021-06-20]. https://arxiv.org/pdf/1901.05803.pdf.

【责任编辑:卓祯雨】